Assignment 2
DCS3101

Kent Odde

October 3, 2020

Contents

Abstract

Q1

There are several ways of visualizing an algorithm, i have chosen to include a visual representation first:

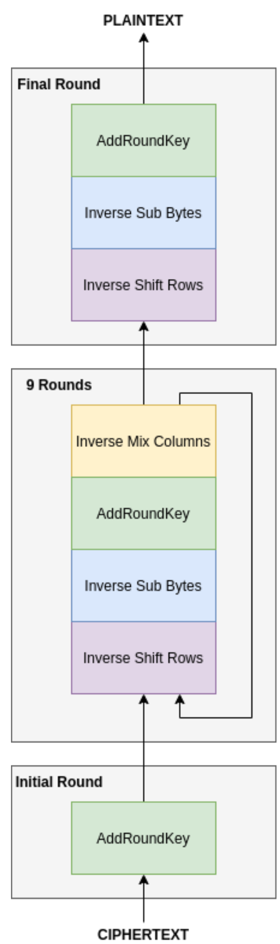

Figure 1: Algorithm for decrypting AES

However, as code is perhaps the most precise way to describe an algorithm, and I already happen to have implemented AES encryption in C++, i decided to also include the decrypt function from that implementation:

```
std::string AES::decrypt128BitMessage(std::string cipherText,
    std::string keyString)
{
   auto key = generateGridFromHexString(keyString);
   auto grid = generateGridFromHexString(cipherText);
   auto expandedKey = generateExpandedKey(key);

   //Initial Round:
   addRoundKey(grid, expandedKey.at(expandedKey.size() - 1));

   //9 Rounds
   for(int i = expandedKey.size() - 2; i > 0; i--)
   {
      invShiftGrid(grid);
      invSubstitute(grid);
      addRoundKey(grid, expandedKey.at(i));
      invMixColumns(grid);
   }

   //Final Round
   invShiftGrid(grid);
   invSubstitute(grid);
   addRoundKey(grid, expandedKey.at(0));

   return gridToHexString(grid);
}
```

Q2

Q3

ECB, CBC, OFB etc. are modes of operation in stream ciphers or when it comes to encrypting data that is longer than the block size given within a certain block cipher algorithm. For instance in AES, a block is 128 bits. If you however want to encrypt more data than this, we use for instance CBC in order to provide additional security across the blocks. The principle is to treat the blocks as a stream.

In this task we have been given these parameters for demonstrating the different modes of operation:

- Plaintext: *DCS-3101*
- IV: *NO*
- Key: *EU*
- Block size: *16 bits*
- Encryption: *XOR*

The block size of sixteen bits means that each block will have two characters, and we will end up with four blocks.

Converted to binary, the plaintext string looks like this:

```
01000100 01000011 01010011 00101101
00110011 00110001 00110000 00110001
```

The block size of sixteen bits means that each block will have two characters, and we will end up with four blocks.

DC	:	01000100 01000011
S-	:	01010011 00101101
31	:	00110011 00110001
01	:	00110000 00110001

The key and IV converted to binary looks like this:

IV	:	01001110 01001111
KEY	:	01000101 01010101

1. ECB

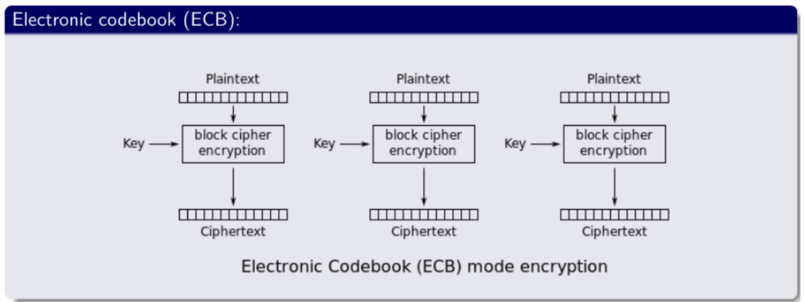

Figure 2: ECB encryption diagram from lecture

ECB is the simplest of the three modes of operation discussed in this assignment, and provide no additional security. The principle is to divide the text into blocks, encrypt them with whatever way we like, and concatenate the ciphertexts produced. This means that if you break one of the blocks, you will be able to break them all. However, an error will not propogate through the blocks.

In practice it will look like this:

01000100 01000011	01010011 00101101	00110011 00110001	00110000 00110001
XOR	**XOR**	**XOR**	**XOR**
01000101 01010101	01000101 01010101	01000101 01010101	01000101 01010101
	=		
00000001 00010110	00010110 01111000	01110110 01100100	01110101 01100100

2. CBC

As the complixity grows in the following operation modes, we will from here on use hexadecimal numbers for increased readability.

In hex, the input data, looks like this:

DC	:	44 43
S-	:	53 2d
31	:	33 31
01	:	30 31

The key and IV converted to binary looks like this:

IV	:	4e 4f
KEY	:	45 55

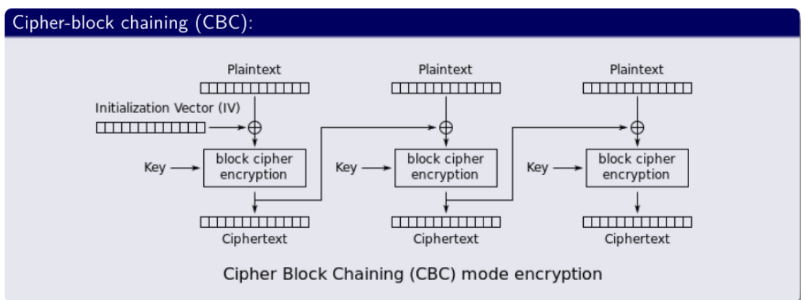

Figure 3: CBC encryption diagram from lecture

CBC stands for cipher-block chaining, and that is exactly what it does. Each plaintext block is XORed with the ciphertext of the previous block. For the first block, we XOR it with an IV (initialization vector). This ensures that each block is dependent on the previous one, and increases the security, as the breaking of one block, does not break the entire message. It does have error propogation however, so an error in a block, wil give an error in all the later blocks.

In practice it looks like this:

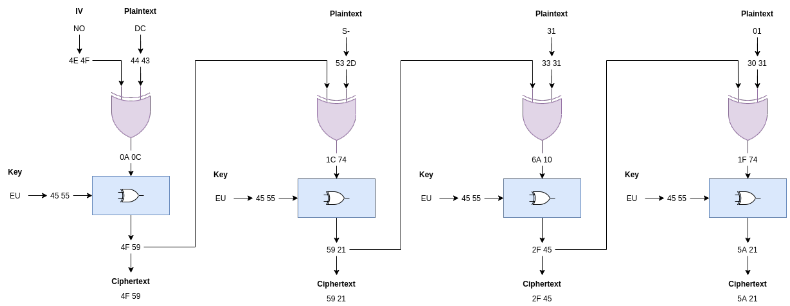

Figure 4: Illustration of CBC

So the encrypted message becomes:

CBC Encrypted: 4F 59 59 21 2F 45 5A 21

3. OFB

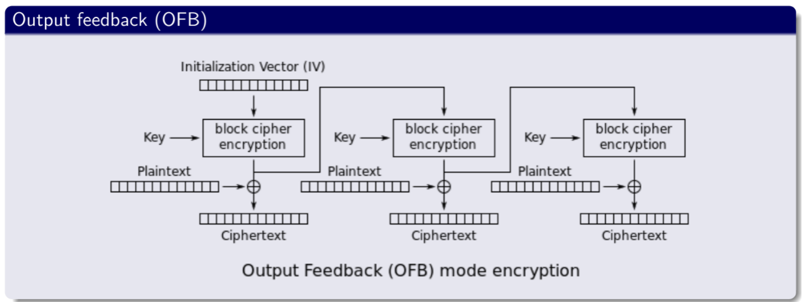

Figure 5: OFB encryption diagram from lecture

OFB (Output feedback), does the block encryption on the IV, rather than on the plaintext. It then XORS the plaintext with the output of the block encryption. The input in the following block encryptions however is the output of the previous block cipher (which does not involve the plaintext). Because of this no blocks are dependent on the plaintext or ciphertext of the previous block, which means that an error will only give an error in the current block (as we will see in the next part).

In practice it looks like this:

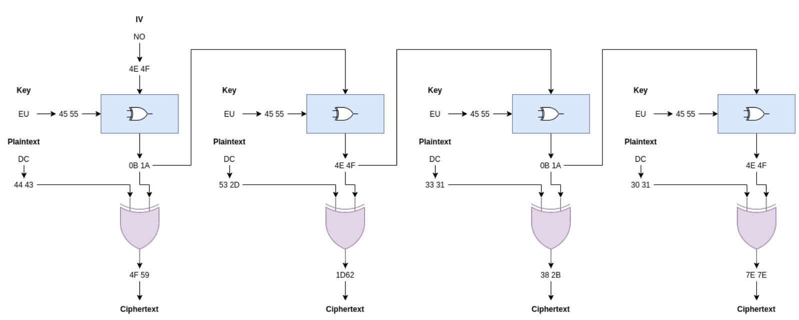

Figure 6: Illustration of OFB

OFB Encrypted: 4F 59 1D 62 38 2B 7E 7E

4. OFB - Decryption

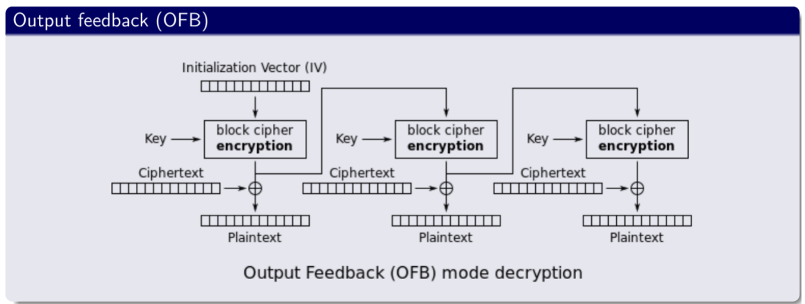

Figure 7: OFB decryption diagram from lecture

In this assignment we will flip the last bit of the first output of the first block from the previos task, and see the effect it has on the decrypted message.

The ciphertext with the flipped bit, looks like this:

Ciphertext: 4F 58 1D 62 38 2B 7E 7E

The fourth hex digit 9, turned into an 8.

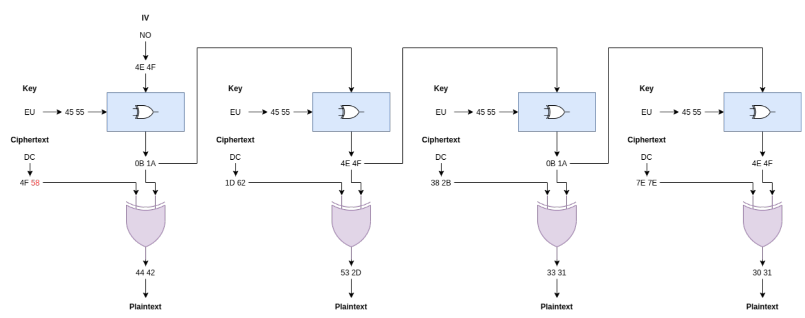

Figure 8: Illustration of OFB decryption with a flipped bit

Decrypted message:

Plaintext: 44 42 53 2D 33 31 30 31

If we take the decrypted message and turn it back into text, it will look like this:

```
Plaintext:    DBS-3101
```

We can now see that the original C, has turned into a B. This corresponds with what we mentioned in the previous task, that the bit error only has an effect on one block.

Comparision

If we compare the results from the three modes:

```
ECB:    01 16 16 78 76 64 75 64
CBC:    4F 59 59 21 2F 45 5A 21
OFB:    4F 59 1D 62 38 2B 7E 7E
```

We of course se that they are drastically different. ECB is without the IV, so it will of course bear the least recemblence to the others. We can also see that while CBC and OFB start of similarly, they quickly divirge.

Q4

Appendices